

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

MICROSOFT CORPORATION,
Petitioner,

v.

THROUGHPUTER, INC.,
Patent Owner.

IPR2022-00574
Patent 10,318,353 B2

Before MICHAEL R. ZECHER, JOHN A. HUDALLA, and
MATTHEW S. MEYERS, *Administrative Patent Judges*.

HUDALLA, *Administrative Patent Judge*.

JUDGMENT
Final Written Decision
Determining Some Challenged Claims Unpatentable
35 U.S.C. § 318(a)

Microsoft Corporation (“Petitioner”) filed a Petition (Paper 2, “Pet.”) requesting an *inter partes* review of claims 1, 2, 8–14, and 21–24 (“the challenged claims”) of U.S. Patent No. 10,318,353 B2 (Ex. 1001, “the

'353 patent"). ThroughPuter, Inc. ("Patent Owner") elected to waive the filing of a preliminary response. Paper 7. We determined that the information presented in the Petition established that there was a reasonable likelihood that Petitioner would prevail with respect to its unpatentability challenges. Pursuant to 35 U.S.C. § 314, we instituted this proceeding on September 19, 2022, as to all challenged claims and all grounds of unpatentability presented in the Petition. Paper 12 ("Dec. on Inst.").

During the course of trial, Patent Owner filed a Patent Owner Response (Paper 19, "PO Resp."), and Petitioner filed a Reply to the Patent Owner Response (Paper 22, "Pet. Reply"). Patent Owner also filed a Sur-reply (Paper 29, "PO Sur-reply"). An oral hearing was held on June 20, 2023, and a transcript of the hearing is included in the record. Paper 33 ("Tr.").

Petitioner filed declarations of Jeffrey S. Chase, Ph.D., with its Petition (Ex. 1003) and its Reply (Ex. 1023). Patent Owner filed a Declaration of Melissa C. Smith, Ph.D., with its Response. Ex. 2002. The parties also filed transcripts of the depositions of Dr. Chase (Exs. 2003, 2006) and Dr. Smith (Ex. 1024).

We have jurisdiction under 35 U.S.C. § 6. This decision is a Final Written Decision under 35 U.S.C. § 318(a) as to the patentability of claims 1, 2, 8–14, and 21–24 of the '353 patent. For the reasons discussed below, Petitioner has demonstrated by a preponderance of the evidence that claims 1 and 2 of the '353 patent are unpatentable. Petitioner, however, has not demonstrated by a preponderance of the evidence that claims 8–14 and 21–24 of the '353 patent are unpatentable.

I. BACKGROUND

A. *Real Parties-in-Interest*

Petitioner identifies Microsoft Corporation as the real party-in-interest. Pet. 88. Patent Owner identifies ThroughPuter, Inc. as the real party-in-interest. Paper 5, 2; Paper 9, 2; Paper 11, 2.

B. *Related Proceedings*

The parties identify the following proceedings related to the '353 patent (Pet. 88; Paper 5, 1; Paper 9, 2; Paper 10, 1; Paper 11, 2):

ThroughPuter, Inc. v. Microsoft Corp., No. 1:21-cv-00397 (E.D. Va. filed Mar. 31, 2021); and

ThroughPuter, Inc. v. Microsoft Corp., No. 3:21-cv-00216 (E.D. Va. filed Mar. 31, 2021); and

ThroughPuter, Inc. v. Microsoft Corp., No. 2:22-cv-00344 (W.D. Wash.; transferred from E.D. Va. Mar. 23, 2022).

We also note that Petitioner has challenged other patents owned by Patent Owner in IPR2022-00527, IPR2022-00528, IPR2022-00757, and IPR2022-00758. We denied institution of *inter partes* review in IPR2022-00757 and IPR2022-00758. Final Written Decisions in IPR2022-00527 and IPR2022-00528 are being issued concurrently with this Decision.

C. *The '353 patent*

The '353 patent is directed to “managing execution of multiple concurrent, multi-task software programs on parallel processing hardware.” Ex. 1001, 1:40–43. Figure 2 of the '353 patent is reproduced below.

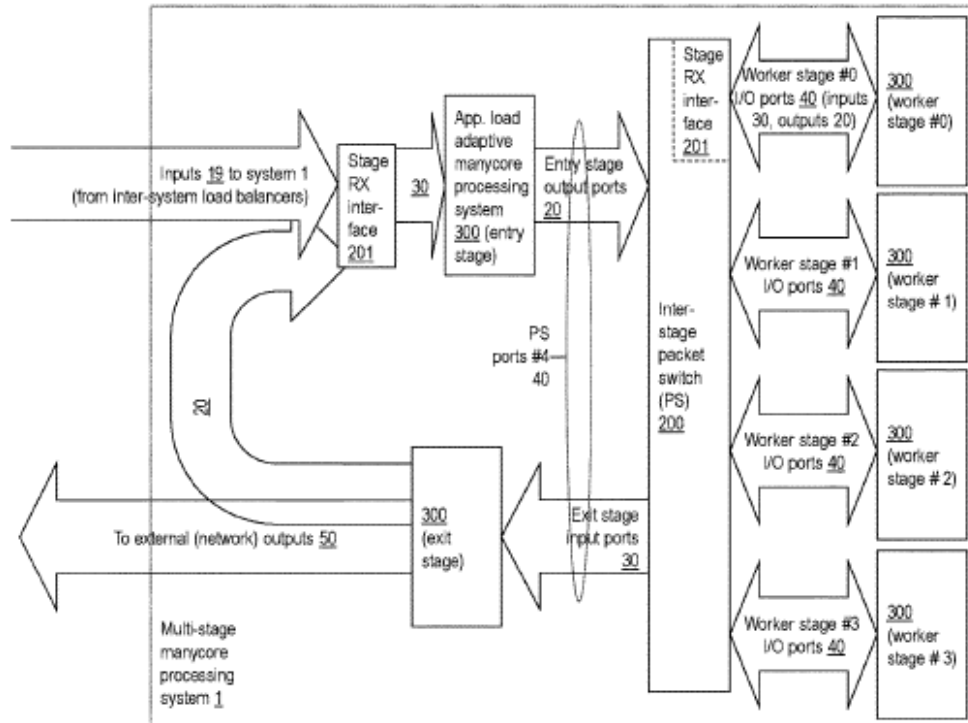


FIG. 2

Figure 2 is a functional block diagram of multi-stage manycore processing system 1, which is shared dynamically among tasks of multiple application programs (apps) and instances (insts) thereof. *Id.* at 5:12–15, 10:23–26. For each of the apps, each task is located at one of the processing stages 300. *Id.* at 10:26–27. Nevertheless, for any given app-inst, copies of same task software can be located at more than one of the processing stages 300 of a given system 1, thus supporting any combination of pipelined and/or parallelized processing. *Id.* at 10:28–35. Hardware logic is implemented for scheduling tasks of application program instances and prioritizing inter-task communications (ITC) among tasks of a given app instance. *Id.* at 10:4–7.

The application that led to the '353 patent claims priority to, *inter alia*, a provisional application that was filed on August 23, 2013 (i.e., the

earliest effective filing date). Ex. 1001, code (60). Petitioner applies this date for qualifying the asserted references as prior art. *See* Pet. 2.

D. Illustrative Claim

Of the challenged claims, claims 1, 8, and 21 are independent. Claim 2 depends from claim 1; claims 9–14 depend directly or indirectly from claim 8; and claims 22–24 depend from claim 21. Claim 1 is illustrative of the challenged claims and recites:

1. [1.P] A system for processing a set of computer program instances, comprising:

[1.1] a plurality of processing stages, at least one of the plurality of processing stages comprising multiple processing cores,

[1.2] wherein, each given task of a plurality of tasks of a given program instance of the set of program instances is hosted at a different stage of the plurality of processing stages as a local task of the given program instance at the respective stage, and

[1.3] for at least one of the multiple processing cores of a given processing stage of the plurality of processing stages, a local task of one of the program instances is assigned as an active task instance for execution for a period of time; and

[1.4] a group of multiplexers each connecting inter-task communications (ITC) data to a respective stage of the plurality of processing stages, wherein at least one multiplexer of the group of multiplexers is a hardware resource dedicated to the local task,

[1.5] wherein the at least one multiplexer is configured to connect ITC data to any processing core of the multiple processing cores to which the local task is assigned for execution for the period of time.

Pet. iv (bracketed numbering by Petitioner); Ex. 1001, 21:18–41.

E. Prior Art

Petitioner relies on the following prior art:

U.S. Patent Application No. 2005/0044344 A1, filed Aug. 21, 2003, published Feb. 24, 2005 (Ex. 1004, “Stevens”);

U.S. Patent No. 6,211,721 B1, filed Dec. 28, 1998, issued Apr. 3, 2001 (Ex. 1005, “Smetana”);

U.S. Patent Application No. 2008/0222640 A1, filed Mar. 7, 2007, published Sept. 11, 2008 (Ex. 1006, “Daly”);

U.S. Patent Application No. 2010/0333099 A1, filed June 30, 2009, published Dec. 30, 2010 (Ex. 1007, “Kupferschmidt”);

U.S. Patent Application No. 2008/0201716 A1, filed Feb. 21, 2007, published Aug. 21, 2008 (Ex. 1008, “Du”); and

U.S. Patent Application No. 2005/0021931 A1, PCT filed Feb. 19, 2002, published Jan. 27, 2005 (Ex. 1009, “Anderson”).

F. The Instituted Grounds

We instituted *inter partes* review of claims 1, 2, 8–14, and 21–24 of the ’353 patent on the following grounds (Dec. on Inst. 38), which are all the grounds presented in the Petition (Pet. 2):

Claims Challenged	35 U.S.C. §	References
1	103 ¹	Stevens, Smetana
2	103	Stevens, Smetana, Daly
8, 13, 21, 24	103	Kupferschmidt, Du, Anderson

¹ The Leahy-Smith America Invents Act (“AIA”), Pub. L. No. 112-29, 125 Stat. 284, 287–88 (2011), amended 35 U.S.C. §§ 102 and 103. Because the ’353 patent claims priority to an application filed after March 16, 2013 (the effective date of the relevant amendments), the AIA versions of §§ 102 and 103 apply. See Ex. 1001, codes (22), (63), (60).

Claims Challenged	35 U.S.C. §	References
9–12, 14, 22, 23	103	Kupferschmidt, Du, Anderson, Daly

II. ANALYSIS

A. *Legal Standards*

A claim is unpatentable under 35 U.S.C. § 103 if the differences between the claimed invention and the prior art are such that the claimed invention, as a whole, would have been obvious before the effective filing date of the claimed invention to a person having ordinary skill in the art to which the claimed invention pertains. *See KSR Int’l Co. v. Teleflex Inc.*, 550 U.S. 398, 406 (2007). The question of obviousness is resolved on the basis of underlying factual determinations, including (1) the scope and content of the prior art; (2) any differences between the claimed subject matter and the prior art; (3) the level of skill in the art; and (4) where in evidence, so-called secondary considerations.² *See Graham v. John Deere Co.*, 383 U.S. 1, 17–18 (1966). We also recognize that prior art references must be “considered together with the knowledge of one of ordinary skill in the pertinent art.” *In re Paulsen*, 30 F.3d 1475, 1480 (Fed. Cir. 1994) (citing *In re Samour*, 571 F.2d 559, 562 (CCPA 1978)).

B. *Level of Ordinary Skill in the Art*

Citing testimony from Dr. Chase, Petitioner contends a person of ordinary skill in the art “would have had a Master’s degree in computer

² The trial record does not include any evidence of secondary considerations of nonobviousness.

science, computer engineering, or a related field, and 2-3 years of practical computer programming or engineering experience, including experience designing or researching parallel processing systems.” Pet. 6–7 (citing Ex. 1003 ¶¶ 18–20). Patent Owner does not dispute Petitioner’s definition of the level of ordinary skill, and Dr. Smith’s testimony is based on a similar definition. *See* Ex. 2002 ¶ 15.

We adopt Petitioner’s definition of the level of ordinary skill in the art and we apply it in our obviousness evaluations below. We are satisfied that this definition comports with the level of skill necessary to understand and implement the teachings of the ’353 patent and the asserted prior art.

C. *Claim Interpretation*

In an *inter partes* review, we construe each claim using the same claim construction standard that would be used to construe the claim in a civil action under 35 U.S.C. [§] 282(b), including construing the claim in accordance with the ordinary and customary meaning of such claim as understood by one of ordinary skill in the art and the prosecution history pertaining to the patent.

37 C.F.R. § 42.100(b) (2021). Accordingly, our claim construction standard is the same as that of a district court. *See id.* Under the standard applied by district courts, claim terms are generally given their plain and ordinary meaning as would have been understood by a person of ordinary skill in the art at the time of the invention and in the context of the entire patent disclosure. *Phillips v. AWH Corp.*, 415 F.3d 1303, 1313 (Fed. Cir. 2005) (en banc). “There are only two exceptions to this general rule: 1) when a patentee sets out a definition and acts as his own lexicographer, or 2) when the patentee disavows the full scope of a claim term either in the

specification or during prosecution.” *Thorner v. Sony Comput. Entm’t Am. LLC*, 669 F.3d 1362, 1365 (Fed. Cir. 2012).

We determine that no aspects of the challenged claims require explicit construction. *See, e.g., Realtime Data, LLC v. Iancu*, 912 F.3d 1368, 1375 (Fed. Cir. 2019) (“The Board is required to construe ‘only those terms . . . that are in controversy, and only to the extent necessary to resolve the controversy.’” (quoting *Vivid Techs., Inc. v. Am. Sci. & Eng’g, Inc.*, 200 F.3d 795, 803 (Fed. Cir. 1999))).

D. Obviousness Ground Based on Stevens and Smetana

Petitioner contends claim 1 would have been obvious over the combination of Stevens and Smetana. Pet. 7–33; Pet. Reply 2–15. Patent Owner disputes Petitioner’s contentions. PO Resp. 4–20; PO Sur-reply 3–14.

1. Stevens

Stevens is a U.S. patent application publication directed to “static and dynamic programming and configuration of an adaptive computing integrated circuit architecture.” Ex. 1004 ¶ 2. Figure 1 of Stevens is reproduced below.

FIG. 1
ADAPTIVE COMPUTING ENGINE (ACE)

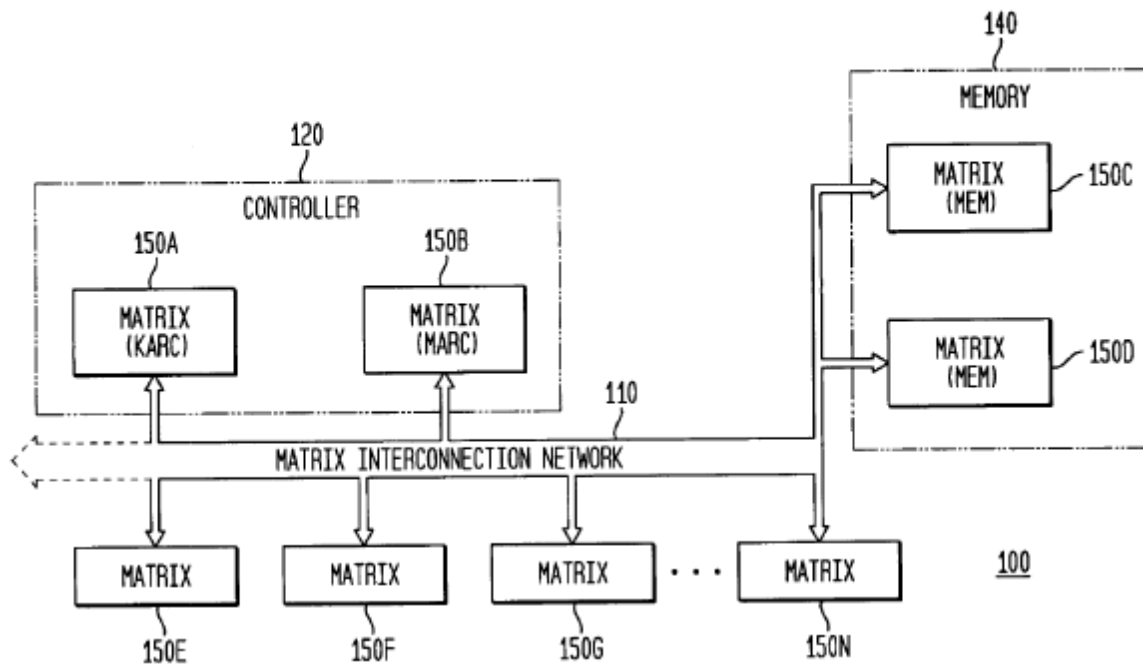


Figure 1 is a block diagram of adaptive computing engine (ACE) 100,³ which is an integrated circuit that is “suitable for digital signal processing, Telematics, and other applications where small hardware footprint, low power consumption and high performance characteristics are highly desirable.” *Id.* ¶¶ 3, 37. ACE 100 includes one or more reconfigurable matrices (or nodes) 150A through 150N and matrix interconnection network 110. *Id.* ¶ 37. Matrices 150A and 150B are configured as controller 120, and matrices 150C and 150D are configured as memory 140. *Id.*

³ Stevens states that an ACE is alternatively called an adaptive computing machine (ACM). Ex. 1004 ¶ 3.

Figure 3 of Stevens is reproduced below.

FIG. 3

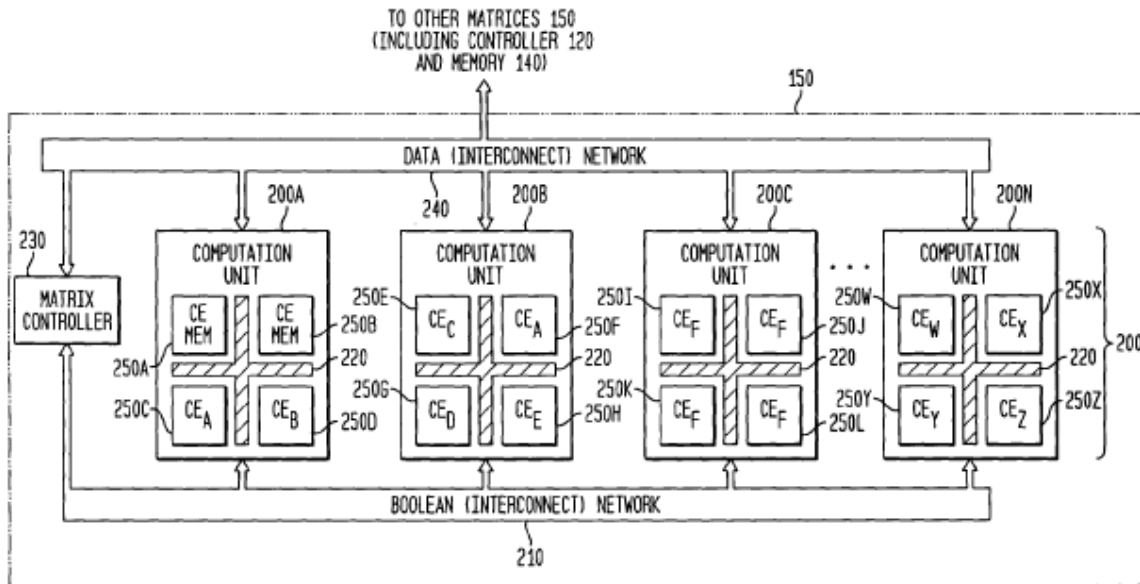


Figure 3 is a block diagram illustrating reconfigurable matrix (or node) 150 with a plurality of computation units 200A through 200N and a plurality of computational elements 250A through 250Z. *Id.* ¶ 54. Matrix 150 also includes matrix controller 230, data interconnect network 240 (which is a portion of matrix interconnect network 110), and Boolean interconnect network 210. *Id.* Each of computational elements 250A through 250Z consists of dedicated, application-specific hardware designed to perform certain fixed tasks. *Id.* ¶ 55.

Figure 4 of Stevens is reproduced below.

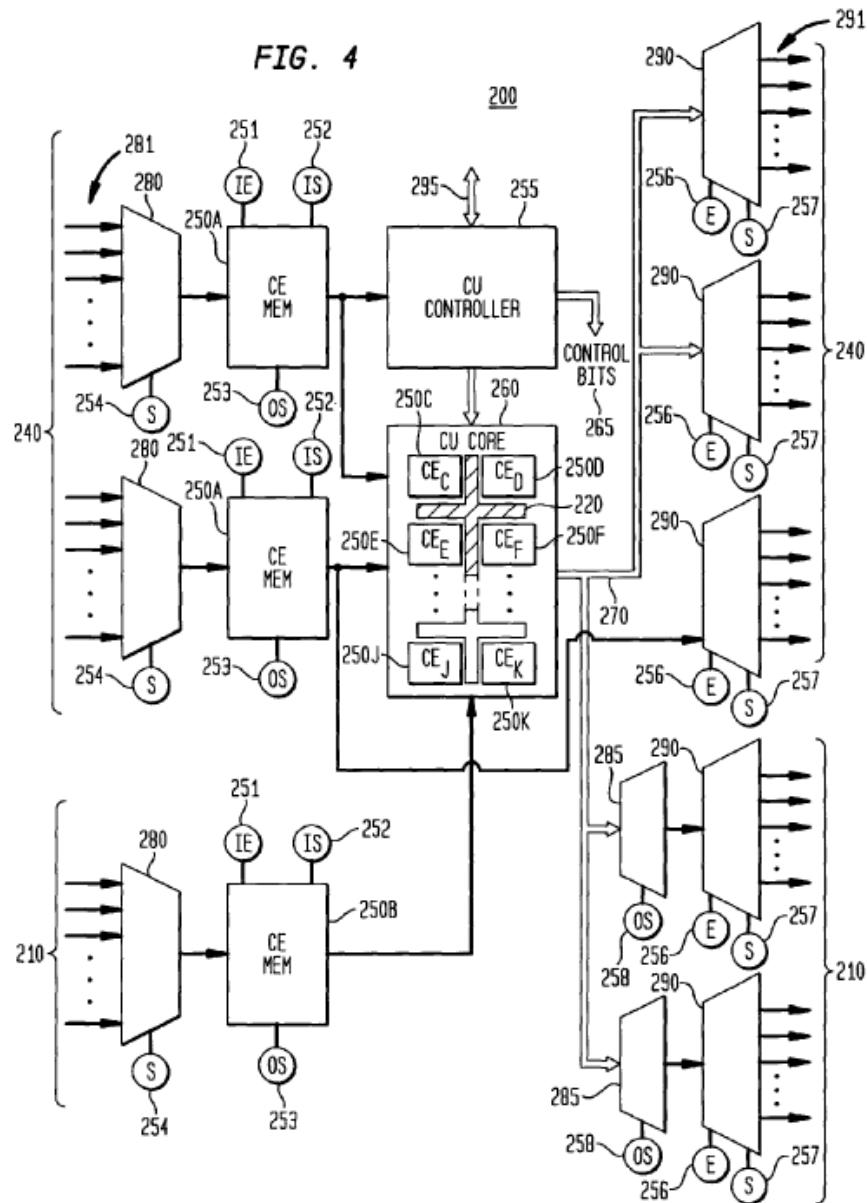


Figure 4 is a block diagram illustrating computation unit 200 of reconfigurable matrix 150. *Id.* ¶ 64. Computation unit 200 includes memory computational elements 250A and 250B and computational unit (CU) core 260. *Id.* In turn, CU core 260 includes algorithmic or finite state machine computational elements 250C through 250K. *Id.* Input multiplexers 280, input lines (or wires) 281, output demultiplexers 285 and

290, and lines (or wires) 291 form data interconnect network 240 and Boolean interconnect network 210. *Id.* ¶ 65.

Stevens also describes a new general-purpose programming language called “SilverC.” *Id.* ¶ 131. Stevens further describes a “SilverC module,” which “acts as a container for program instructions and data that will be used to perform some computation on some hardware platform[s], such as a node within the ACE (ACM) 100.” *Id.* ¶ 133.

Petitioner contends that Stevens qualifies as prior art under 35 U.S.C. § 102(a)(1) and (a)(2) based on its publication date. Pet. 2. Patent Owner does not dispute the prior art status of Stevens. We determine that Stevens qualifies as prior art under 35 U.S.C. § 102(a)(1) and (a)(2) because Stevens’s publication date of February 24, 2005, is before the earliest effective filing date of the challenged claims, which is August 23, 2013. Ex. 1001, code (60); Ex. 1004, code (43).

2. *Smetana*

Smetana is a U.S. patent directed to “a digital multiplexer with a short data to output propagation delay and low power consumption.” Ex. 1005, 1:7–10. Smetana notes the desirability of “transmit[ting] data from an input of a multiplexer to the output of the multiplexer with as little delay as possible, and with as little power dissipation as possible.” *Id.* at 1:15–18.

Petitioner contends that Smetana qualifies as prior art under 35 U.S.C. § 102(a)(1) and (a)(2) based on its issue date. Pet. 2. Patent Owner does not dispute the prior art status of Smetana. We determine that Smetana qualifies as prior art under 35 U.S.C. § 102(a)(1) and (a)(2) because Smetana’s issue date of April 3, 2001, is before the earliest effective filing date of the

challenged claims, which is August 23, 2013. Ex. 1001, code (60);
Ex. 1005, code (45).

3. *Claim 1*

a. Preamble [1.P]

The preamble of claim 1 recites “[a] system for processing a set of computer program instances.” Ex. 1001, 21:18–19. Petitioner cites Stevens’s teaching of an adaptive computer engine (ACE) for teaching the recited “system.” Pet. 10 (citing Ex. 1004 ¶¶ 3–8, 35–37, 51, Figs. 1, 10, 15). For the recited “set of computer program instances,” Petitioner contends that Stevens’s ACE can support as few as one program instance, but Stevens also teaches that the ACE can support multiple program instances, such as applications for a music player, a pager, and a personal digital assistant. *Id.* at 10–11 (citing Ex. 1003 ¶¶ 78–80; Ex. 1004 ¶¶ 51, 80).

Neither party addresses whether the preamble is limiting. Because Petitioner shows persuasively that Stevens teaches the preamble (*see, e.g.*, Ex. 1004 ¶¶ 3–8, 35–37, 51, 80, Fig. 1), we need not determine whether the preamble is limiting. *See Realtime Data*, 912 F.3d at 1375.

b. Limitation [1.1]

Claim 1 further recites “a plurality of processing stages, at least one of the plurality of processing stages comprising multiple processing cores” (Limitation [1.1]). Ex. 1001, 21:20–22. Petitioner maps the recited “processing stages” to Stevens’s matrices 150 and matrices 800. Pet. 11–13 (citing Ex. 1004 ¶¶ 37–49, 62, 63, 79, 80, 101–110, Figs. 1, 10). Petitioner

explains that the “[m]atrices in [Stevens’s] ACE are configurable such that a given matrix can be configured or reconfigured to implement a specific task of a computer program in hardware” or may be “application specific, having a fixed architecture with a corresponding fixed function.” *Id.* at 12–13 (citing Ex. 1004 ¶¶ 43, 48, 62, 63). For the recited “processing cores,” Petitioner contends that computation units 200 within Stevens’s matrices 150 and matrices 800 each include “a respective set of computational elements 250 that ‘form[] a computational unit (“CU”) core 260.’” *Id.* at 14 (quoting Ex. 1004 ¶ 64) (citing *id.* ¶¶ 64–68, 138, Fig. 4) (alteration by Petitioner). Petitioner also explains that “Stevens’s second matrix architecture (e.g., matrices 800) similarly includes ‘a plurality of computation units 200’ (multiple processing cores) within an adaptive execution unit 840.” *Id.* at 14 n.2 (citing Ex. 1003 ¶ 96; Ex. 1004 ¶¶ 79–80, 89–100, Fig. 10).

Patent Owner argues that “each matrix in Stevens includes a single adaptive execution unit to which tasks are assigned for execution one task at a time.” PO Resp. 5 (citing Ex. 2002 ¶ 28). Patent Owner’s arguments hinge on Petitioner’s mapping of the recited “task” to a process in Stevens’s SilverC module⁴ (*see* Pet. 15–16), which Patent Owner in turn relates to Stevens’s matrices. PO Resp. 8 (citing Ex. 2002 ¶ 38). Patent Owner then cites Stevens’s teaching that “SilverC modules are code containers that are mapped (by a compiler) to a *single* ‘execution unit.’” *Id.* (emphasis by Patent Owner) (quoting Ex. 1004 ¶ 138). In support of this argument, Patent Owner cites Stevens’s teachings about how only a single task at a time is

⁴ We discuss this mapping below. *See infra* § II.D.3.c.

executed within adaptive execution unit 840. *Id.* at 9–10 (citing Ex. 1004 ¶¶ 94–96; Ex. 2002 ¶¶ 34).

Based on the fully developed trial record, we do not agree with Patent Owner’s arguments. First, we do not read Petitioner’s unpatentability contentions to be limited to Stevens’s embodiment that includes an adaptive execution unit. Rather, with reference to Stevens’s Figure 3, Petitioner also cites Stevens’s matrices 150 for teaching the recited “plurality of processing stages” and Stevens’s computation units 200 and/or CU cores 260 for teaching the recited “plurality of processing cores.” *See* Pet. 13–14; Ex. 1004 ¶¶ 43 (“The various reconfigurable matrices 150 each generally contain a different or varied mix of adaptive and reconfigurable computational (or computation) units (200).”), 64 (“[A] computation unit 200 typically includes a plurality of diverse, heterogeneous and fixed computational elements 250, such as a plurality of memory computational elements 250A and 250B, and forming a computational unit (‘CU’) core 260, a plurality of algorithmic or finite state machine computational elements 250C through 250K.”). We are persuaded by these contentions tied to matrices 150 irrespective of Petitioner’s reliance on SilverC processes for teaching the recited “tasks” in claim 1. *See In re Applied Materials, Inc.*, 692 F.3d 1289, 1298 (Fed. Cir. 2012) (“A reference must be considered for everything that it teaches, not simply the described invention or a preferred embodiment.”).

Second, even if we were to agree with Patent Owner that Petitioner’s reliance on SilverC processes locks Petitioner in to Stevens’s embodiment including adaptive execution unit 840, Petitioner directly addresses this possibility in the Petition. Specifically, Petitioner states that “Stevens’s

second matrix architecture (e.g., matrices 800) similarly includes ‘a plurality of computation units 200’ (***multiple processing cores***) within an adaptive execution unit 840.” Pet. 14 n.2 (citing Ex. 1003 ¶ 96; Ex. 1004 ¶¶ 79–80, 89–100, Fig. 10); *see also id.* at 12 (citing Ex. 1004 ¶¶ 79–80, 101–110, Fig. 10). As acknowledged by Patent Owner (*see* PO Resp. 8), Stevens supports Petitioner’s analysis: “Each adaptive execution unit 840 . . . is comprised of a plurality of computation units 200, which are in turn further comprised of a plurality of computational elements 250.” Ex. 1004 ¶ 80. Thus, we disagree with Patent Owner’s unduly narrow view of an adaptive execution unit as being limited to a single processing core, which is contradicted by Stevens itself. We find that Stevens’s matrices teach the recited “processing stages” (*see, e.g.*, Ex. 1004 ¶¶ 37–49, 62, 63, 79, 80, 101–110, Figs. 1, 10) and Stevens’s computational units and computation unit cores teach the recited “processing cores” (*see, e.g., id.* ¶¶ 64–68, 138, Fig. 4).

Patent Owner also incorrectly suggests that Stevens’s teachings on cooperative multitasking undermine Petitioner’s analysis. In particular, Patent Owner contends that these teachings (*see* Ex. 1004 ¶ 151) imply that only one task can be active at a time in each matrix, which means that there is no need for multiple processing cores per matrix. PO Resp. 10 (citing Ex. 2002 ¶¶ 42–43; Ex. 2003, 43:13–24). Patent Owner concludes that “Stevens thus includes a single processing core per matrix to which tasks are assigned.” *Id.* at 11. Yet, as discussed above, Stevens teaches multiple computation units and CU cores within a matrix. Further, claim 1 does not require multiple tasks to be executed at once. Patent Owner and its declarant have acknowledged as much. *See* Tr. 22:14–17; Ex. 1024, 64:14–18. Thus,

we do not agree with Patent Owner’s arguments related to Stevens’s cooperative multitasking.

For these reasons, we find that Stevens teaches Limitation [1.1].

c. Limitation [1.2]

Claim 1 further recites that “each given task of a plurality of tasks of a given program instance of the set of program instances is hosted at a different stage of the plurality of processing stages as a local task of the given program instance at the respective stage” (Limitation [1.2]). Ex. 1001, 21:23–27. Petitioner cites Stevens’s teaching of “SilverC” program modules that each consist of processes, with each process corresponding to a task to be performed on a selected node. Pet. 15–16 (citing Ex. 1004 ¶¶ 97, 134, 190). Petitioner contends that two or more of Stevens’s matrices 150/800 can each host a respective task of a program instance, and that each task assigned to a given matrix is a “local task.” *Id.* at 16 (citing Ex. 1003 ¶ 92; Ex. 1004 ¶ 113).⁵

Patent Owner does not dispute Petitioner’s contentions.⁶ We find that processes within Stevens’s SilverC module teach the recited “tasks” that are performed at matrices 150/800. *See, e.g.*, Ex. 1003 ¶ 92; Ex. 1004 ¶¶ 97,

⁵ In its Reply, Petitioner puts forth a new alternative mapping of the recited “task” to “algorithmic elements” in Stevens. Pet. Reply 13–15. Patent Owner points out certain deficiencies about Petitioner’s new mapping. PO Sur-reply 1–14. We do not reach the issues pertaining to Petitioner’s new mapping from the Reply because we are persuaded by Petitioner’s original contentions from the Petition.

⁶ As discussed above and below, Patent Owner does dispute other contentions based on Petitioner’s mapping of “tasks” to processes in Stevens’s SilverC module. *See supra* § II.D.3.b; *infra* §§ II.D.3.d.–g.

113, 134, 190. We also find that Stevens’s matrices 150/800 (i.e., the recited “processing stages”) host tasks associated with program instances. *See, e.g.*, Ex. 1003 ¶ 92; Ex. 1004 ¶¶ 97, 113, 134, 190.

d. Limitation [1.3]

Claim 1 further recites that “for at least one of the multiple processing cores of a given processing stage of the plurality of processing stages, a local task of one of the program instances is assigned as an active task instance for execution for a period of time” (Limitation [1.3]). Ex. 1001, 21:28–32. Petitioner cites Stevens’s teachings of changing task states from “suspend,” “idle,” or “ready-to-run” to “running,” and contends that this teaches the recited assignment of an “active task instance” for execution over a period of time. Pet. 18 (citing Ex. 1003 ¶ 97; Ex. 1004 ¶¶ 95–97, 99). Petitioner further contends that, when a task runs on corresponding matrix 150/800, the task is assigned as an “active task instance” “for at least one of the multiple processing cores” of that matrix because “the local task is activated on the computation units 200A–N (and their corresponding CU cores 260).” *Id.* at 18–19 (citing Ex. 1003 ¶ 98; Ex. 1004 ¶¶ 53–54, 64–68, 80, Figs. 3, 4, 10).

Patent Owner argues that Petitioner does not establish “that tasks are assigned to each of the four computation units of Stevens (or that the computation units are even capable of executing a task).” PO Resp. 16 (citing Ex. 2002 ¶¶ 51–53). In support of this argument, Patent Owner contends “[t]he level of control needed to assign different tasks to different computation units is simply not present in Stevens.” *Id.* at 18 (citing Ex. 2002 ¶ 52).

Based on the fully developed trial record, we do not agree with Patent Owner's arguments, which are again rooted in a rigid notion of how Stevens's adaptive execution unit relates to a matrix. *See, e.g.*, PO Resp. 20 ("[T]asks are assigned to a single adaptive execution unit, which executes only a single task at a time."); *see also supra* note 4. Contrary to Patent Owner's arguments, Petitioner shows persuasively that "[t]he local task is assigned as an 'active task instance' . . . each time the task is executed on its assigned matrix 150/800." Pet. 18 (citing Ex. 1004 ¶¶ 95–99). Petitioner further explains that, "[b]y executing the local task hosted on a matrix 150/800, the local task is activated on the computation units 200A-N (and their corresponding CU cores 260)." *Id.* at 18–19; Ex. 1002 ¶ 98.

Petitioner's explanation still holds true when considering the embodiment of Stevens where matrix 800 includes adaptive execution unit 840. We agree with Petitioner that "[d]ispatching or assigning a task to an AEU 840 in Stevens further includes assigning the task to one or more computational units 200 and corresponding CU cores 260 within AEU 840." Pet. Reply 8 (citing Ex. 1004 ¶¶ 79–80, 95–100, Figs. 3–4, 10). This is consistent with Stevens's teachings of matrix 800 including adaptive execution unit 840, which, in turn, is comprised of computation units 200 and computational elements 250, as we have discussed above. *See* Ex. 1004 ¶¶ 79–80. And, also as discussed above, we disagree with any suggestion by Patent Owner that multiple computation units and/or CU cores must operate simultaneously in order to meet the claim language. *See supra* § II.D.3.b.

Thus, we find that Stevens teaches that a local task is assigned as an "active task instance" that is executed by processing cores. *See, e.g.*, Ex. 1003 ¶¶ 97–98; Ex. 1004 ¶¶ 53–54, 64–68, 80, 95–97, 99, Figs. 3, 4, 10.

e. Limitation [1.4]

Claim 1 further recites “a group of multiplexers each connecting inter-task communications (ITC) data to a respective stage of the plurality of processing stages” (Limitation [1.4]). Ex. 1001, 21:33–35. Petitioner cites Stevens’s teaching that the ACE includes matrix interconnection network (MIN) 110, which “provides communication paths between and within matrices 150/800.” Pet. 19 (citing Ex. 1003 ¶¶ 100–110; Ex. 1004 ¶¶ 41–42, 85, Figs. 1, 3, 11–13). Petitioner contends this interconnection network “extends into the matrices 150/800” (i.e., the recited “processing stages”) via data interconnect network 240 and Boolean interconnect network 210, which “provide connections between matrix controller 230 and each of multiple computation units 200A–N.” *Id.* at 20–21 (citing Ex. 1003 ¶¶ 100–102; Ex. 1004 ¶¶ 41, 54, Fig. 3). Petitioner further cites Stevens’s teaching that interconnection network 110 includes “combinations of routing elements and multiplexing elements . . . at differing levels within the system.” *Id.* at 22 (quoting Ex. 1004 ¶ 42). Thus, with respect to the recited “group of multiplexers,” Petitioner cites Stevens’s input multiplexers 280 for computation unit 200, which Petitioner contends are a part of conceptual data network 240 and Boolean interconnect network 210. *Id.* at 22–23 (citing Ex. 1004 ¶ 65, Fig. 4). Petitioner contends that data communicated among matrices 150/800 constitutes the recited “inter-task communications (ITC) data.” *Id.* at 23–24 (citing Ex. 1003 ¶ 105; Ex. 1004 ¶¶ 89–91, 95, 113–119, 190, 202–206).

Patent Owner makes arguments related to limitation [1.4], but they only pertain to Petitioner’s new alternative mapping of the recited “task” from Petitioner’s Reply, as Patent Owner acknowledges. *See*

PO Sur-reply 10–12; *see also supra* note 4. Given that we have analyzed and are persuaded by Petitioner’s original mapping of “task” from the Petition, Patent Owner’s arguments do not apply. Patent Owner does not otherwise dispute Petitioner’s analysis of limitation [1.4]. We find that Stevens teaches multiplexers in matrices 150/800 (i.e., “processing stages”) that are used as part of various exchanges of data (i.e., “inter-task communications (ITC) data”) among the matrices. *See, e.g.*, Ex. 1003 ¶ 105; Ex. 1004 ¶¶ 42, 65, 89–91, 95, 113–119, 190, 202–206, Fig. 4.

f. Limitation [1.5] and Rationale for Combining Smetana with Stevens

Claim 1 further recites that “at least one multiplexer of the group of multiplexers is a hardware resource dedicated to the local task” (limitation [1.5]). Ex. 1001, 21:35–37. Based on Stevens’s teaching that the ACE is an integrated circuit “hardware platform,” Petitioner contends that an ordinarily skilled artisan “would have understood the components of the ACE—such as multiplexers 280 (or 955 under the alternative mapping)—to be hardware resources as recited in claim 1.” Pet. 27 (citing Ex. 1003 ¶¶ 74–75, 111; Ex. 1004 ¶¶ 2–3, 5). Nevertheless, to the extent Stevens does not expressly describe its multiplexers as a hardware resource, Petitioner cites Smetana’s hardware multiplexers. *Id.* at 28 (citing Ex. 1005, 2:35–37, 2:60–4:3, Fig. 2). Petitioner contends that such hardware multiplexers “were exceedingly common electronic components for decades before the ’353 patent,” and that an ordinarily skilled artisan would have implemented Stevens’s multiplexers 280 and 955 as hardware multiplexers to realize known benefits, such as “reducing propagation delay and power consumption.” *Id.* (citing Ex. 1003 ¶¶ 74–75; Ex. 1005, 1:13–18).

Petitioner additionally contends that Stevens “contemplates scenarios in which a 1:1 correspondence is established between a task and a node 150/800 such that the node is ‘dedicated’ to that task as the only task hosted locally on that node.” *Id.* at 28–29 (citing Ex. 1003 ¶¶ 112–113; Ex. 1004 ¶¶ 95, 97, 133–134, 137, 190).

Patent Owner makes arguments related to limitation [1.5], but they only pertain to Petitioner’s new alternative mapping of the recited “task” from Petitioner’s Reply, as Patent Owner acknowledges. *See* PO Sur-reply 10–12; *see also supra* note 4. Given that we have analyzed and are persuaded by Petitioner’s original mapping of “task” from the Petition, these arguments do not apply. Patent Owner does not otherwise dispute Petitioner’s analysis of limitation [1.5]. Nor does Patent Owner dispute Petitioner’s rationale for combining Smetana with Stevens.

We find that Smetana teaches the implementation of multiplexers in hardware. *See, e.g.,* Ex. 1005, 2:35–37, 2:60–4:3, Fig. 2. We also find that Stevens teaches matrices/nodes 150/800 that are sometimes dedicated to a particular task. *See, e.g.,* Ex. 1003 ¶¶ 112–113; Ex. 1004 ¶¶ 95, 97, 133–134, 137, 190. Furthermore, Petitioner persuasively shows that an ordinarily skilled artisan would have implemented Stevens’s multiplexers 280 and 955 as hardware multiplexers based on the teachings of Smetana to obtain the benefits of reduced propagation delay and reduced power consumption. *See, e.g.,* Ex. 1003 ¶¶ 74–75; Ex. 1005, 1:13–18. We note that Stevens already describes the ACE as an integrated circuit hardware platform (*see* Ex. 1004 ¶¶ 2–3, 5), which supports that an ordinarily skilled artisan would have expected success in implementing the components of the ACE, such as

Stevens's multiplexers, as hardware resources based on Smetana's teachings. *See* Pet. 27 (citing, *inter alia*, Ex. 1003 ¶¶ 74–75, 111).

g. Limitation [1.6]

Claim 1 further recites that “the at least one multiplexer is configured to connect ITC data to any processing core of the multiple processing cores to which the local task is assigned for execution for the period of time” (limitation [1.6]). Ex. 1001, 21:38–41. Petitioner recapitulates its contention discussed above that “local multiplexers 280 on a given node 150/800 . . . are configured to connect ITC data . . . for processing by CU cores 260 . . . of computation units 200A-N.” Pet. 30–31 (citing Ex. 1003 ¶¶ 100–110, 116–121; Ex. 1004 ¶¶ 64–65, Figs. 3–4). In light of this, Petitioner contends that, in Stevens, “the set of multiplexers 280 across all computational units 200A-N on an individual node 150/800 . . . collectively enable connection of data inputs to any and all of the cores of the node 150/800.” *Id.* at 32–33 (citing Ex. 1003 ¶¶ 117–118).

Patent Owner makes arguments related to limitation [1.6], but they only pertain to Petitioner's new mapping of the recited “task” from Petitioner's Reply, as Patent Owner acknowledges. *See* PO Sur-reply 10–12; *see also supra* note 4. Given that we have analyzed and are persuaded by Petitioner's original mapping of “task” from the Petition, Patent Owner's arguments do not apply. Patent Owner does not otherwise dispute Petitioner's analysis of limitation [1.6]. We find that Stevens teaches interconnected computation unit cores, and that ITC data may travel via connected multiplexers to the cores for local task execution. *See, e.g.*, Ex. 1003 ¶¶ 100–110, 116–121; Ex. 1004 ¶¶ 64–65, Figs. 3–4.

h. Conclusion for Claim 1

Petitioner has persuasively shown that the combination of Stevens and Smetana teaches all limitations of claim 1. Petitioner also has put forth persuasive reasons for combining these references. Thus, regarding claim 1, we determine that Petitioner has shown, by a preponderance of the evidence, that the claimed invention as a whole would have been obvious over the combination of Stevens and Smetana.

E. Obviousness Ground Based on Stevens, Smetana, and Daly

Petitioner contends claim 2 would have been obvious over the combination of Stevens, Smetana, and Daly. Pet. 33–37; Pet. Reply 2–15. Patent Owner disputes Petitioner’s contentions. PO Resp. 4–20; PO Sur-reply 3–14.

1. Daly

Daly is a U.S. patent application publication directed to “task scheduling in computing systems.” Ex. 1006 ¶ 1. Figure 1 of Daly is reproduced below.

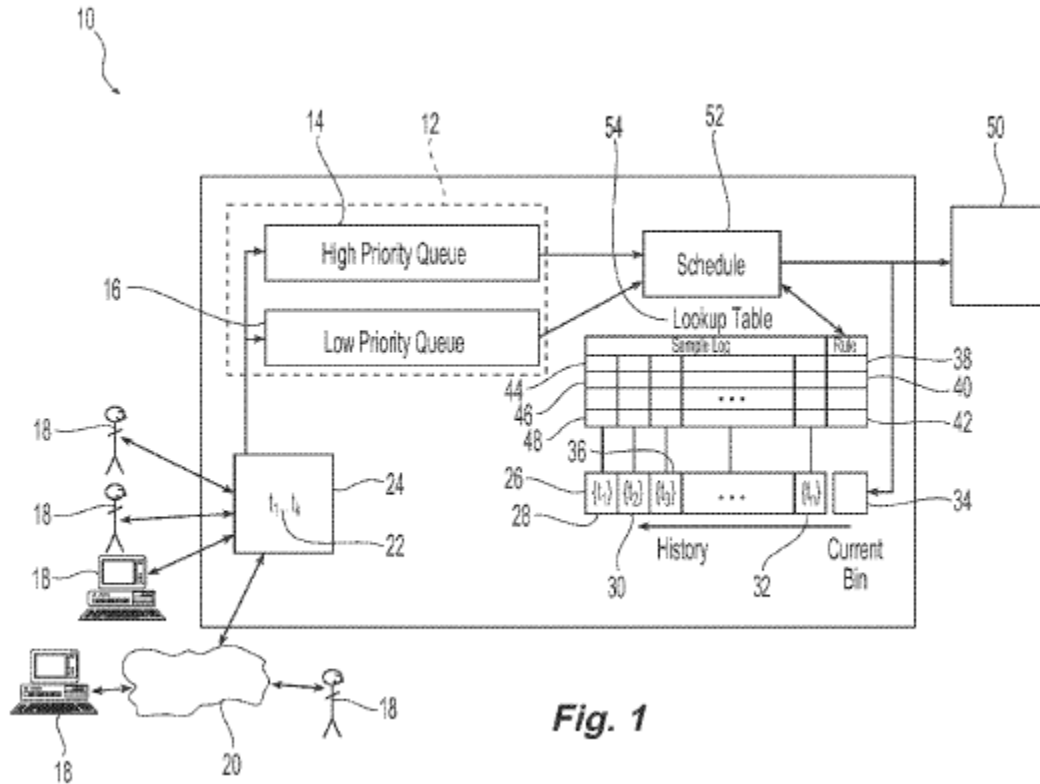


Figure 1 depicts system 10 for use in scheduling tasks. *Id.* ¶ 16.

System 10 includes high priority queue 14, low priority queue 16, and buffer 24 to receive and hold submitted task requests. *Id.* ¶ 17. System 10 is capable of evaluating each task request in buffer 24, determining the proper priority level for the task request, and communicating the task request to the appropriate queue. *Id.* Queues are differentiated based upon characteristics of the task requests to be scheduled, such as a queue for each user (or source) submitting task requests and a queue for a particular type or class of task request. *Id.*

Petitioner contends that Daly qualifies as prior art under 35 U.S.C. § 102(a)(1) and (a)(2) based on its publication date. Pet. 2. Patent Owner does not dispute the prior art status of Daly. We determine that Daly qualifies as prior art under 35 U.S.C. § 102(a)(1) and (a)(2) because Daly's publication date of September 11, 2008, is before the earliest effective filing

date of the challenged claims, which is August 23, 2013. Ex. 1001, code (60); Ex. 1006, code (43).

2. *Claim 2*

Claim 2 depends from claim 1 and recites that the system further comprises

a buffer located at a given one of the plurality of processing stages, the buffer provided to a first task of the plurality of tasks of the given program instance and being specific to another task of the plurality of tasks of the given program instance, wherein the buffer buffers the ITC data originating from said another task to the first task.

Ex. 1001, 21:42–49. Petitioner cites Daly’s teaching of “a plurality of queues 12 to hold the task requests,” which “are differentiated based upon one or more characteristics of the task requests to be scheduled.” Pet. 35–36 (quoting Ex. 1006 ¶¶ 17–18). Petitioner contends that

[i]t would have been obvious to apply Daly’s teachings for source-specific input queues in combination with Stevens and Smetana such that a consumer task (***first task***) of a given program instance hosted on a given matrix 150/800 (***processing stage***) would include source-specific inpipes for buffering or queueing data from one or more upstream sources (producer tasks).

Pet. 36 (citing Ex. 1003 ¶ 129). Petitioner explains that “a first inpipe to a downstream task (***first task***) would provide a buffer specific to a particular upstream task (***another task of the plurality of tasks of the given program instance***).” *Id.*

Building on the Stevens–Smetana combination discussed above, Petitioner contends an ordinarily skilled artisan would have “sought to provide source-specific inpipes for queueing inputs” in order “to more readily

differentiate among inputs from different source tasks” so that “the predictable benefits of such discrimination could be enjoyed in a manner similar to Daly’s system.” *Id.* at 37 (citing Ex. 1003 ¶ 129). Petitioner touts Daly’s teaching of improved scheduling via “source-specific queueing,” which “allows inputs from each source to be prioritized relative to each other based on relative queue priorities.” *Id.* (citing Ex. 1006 ¶¶ 16–18). According to Petitioner, “[s]ource-specific inpipes for a task in [Stevens’s] ACE system would similarly allow the hardware task manager on a matrix to better prioritize the order of inputs processed by a task according to the priority of the source from which the input was produced.” *Id.* (citing Ex. 1003 ¶ 129; Ex. 1004 ¶¶ 93–95, 120–130). Petitioner also notes that Stevens expressly contemplates inpipes for queuing inputs “to permit communication among related tasks of a program on two or more matrices or nodes 150/800 of an ACE,” and that Stevens’s modules may include one or more pipes per process or task. *Id.* (citing Ex. 1004 ¶¶ 133–135, 153–166, 202–214).

Patent Owner disputes neither Petitioner’s analysis of claim 2 nor Petitioner’s rationale for combining Daly with Stevens and Smetana. Otherwise, Patent Owner relies on the same arguments we have discussed with respect to the Stevens–Smetana ground. *See supra* § II.D.3. We find that Daly’s source-specific input queues teach the recited “buffer” of claim 2. *See, e.g.*, Ex. 1006 ¶¶ 16–18. We also are persuaded that an ordinarily skilled artisan would have applied Daly’s input queues to Stevens’s matrices so that a program instance at a matrix would include source-specific inpipes for buffering data from upstream tasks. *See, e.g.*, Ex. 1003 ¶ 129. Stevens’s express teaching of multiple pipes per process module (Ex. 1004 ¶ 133)

supports that an ordinarily skilled artisan would have successfully implemented Daly's source-specific input queues in the Stevens–Smetana system. Thus, regarding claim 2, we determine that Petitioner has shown, by a preponderance of the evidence, that the claimed invention as a whole would have been obvious over the combination of Stevens, Smetana, and Daly.

F. Obviousness Ground Based on Kupferschmidt, Du, and Anderson

Petitioner contends claims 8, 13, 21, and 24 would have been obvious over the combination of Kupferschmidt, Du, and Anderson. Pet. 38–76; Pet. Reply 15–26. Patent Owner disputes Petitioner's contentions. PO Resp. 21–40; PO Sur-reply 14–27.

1. Kupferschmidt

Kupferschmidt is a U.S. patent application publication directed to inter-communication of hardware threads in a processor architecture. Ex. 1007 ¶ 1. A hardware thread is at least a portion of a processor configured as an independent unit of execution to process at least a portion of a task. *Id.* ¶ 26. Hardware threads may be configured with inboxes to receive messages for processing a task. *Id.* ¶ 7. Each hardware thread is configured to receive at least one message carrying data to process the workload through a respective inbox from among a plurality of inboxes. *Id.* ¶ 8. Hardware threads may have at least two inboxes: a priority inbox (also known as a “priority message buffer”) and a second inbox for messages not associated with a priority. *Id.* ¶ 29. Each message is associated with a priority, and the message with the highest priority among those received at

an inbox may be selected for processing. *Id.* ¶ 8. Messages may also be selected based on the message having the earliest time stamp. *Id.*

Figure 2 of Kupferschmidt is reproduced below.

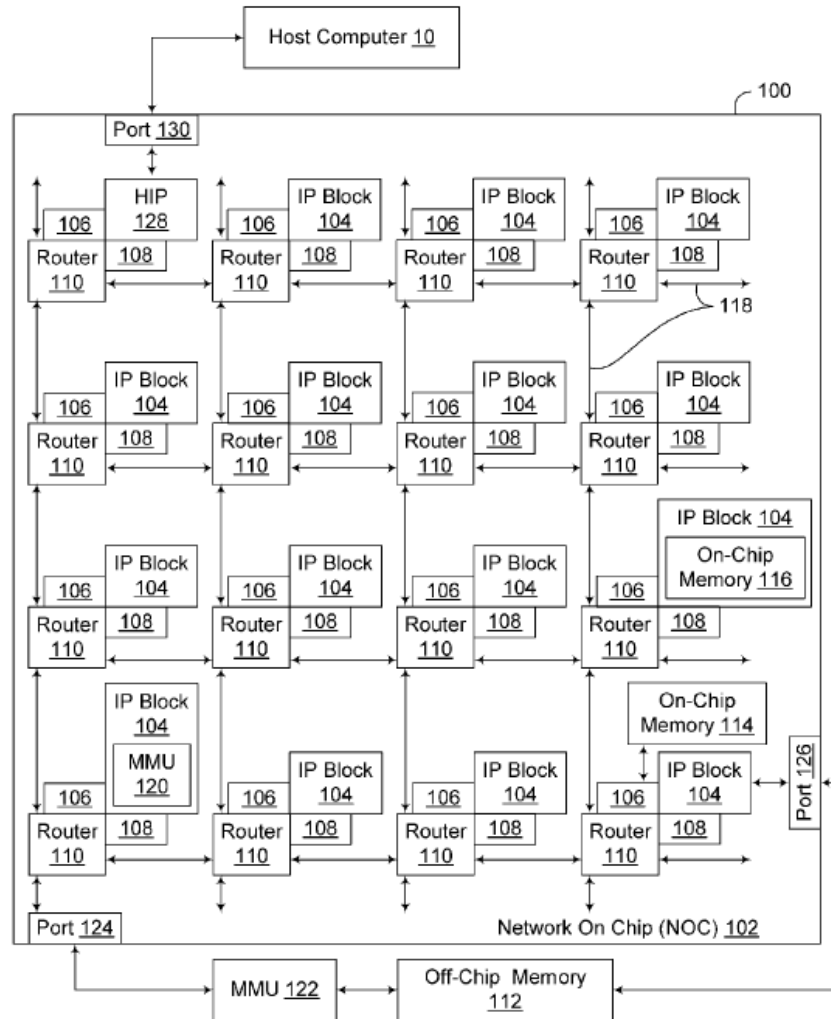


FIG. 2

Figure 2 is a functional block diagram of Network On Chip (NOC) integrated circuit device 102, such as a video adapter or a coprocessor, that is implemented on integrated circuit/chip 100. Ex. 1007 ¶¶ 40, 44. NOC 102 includes integrated processor (IP) blocks 104, routers 110, memory communications controllers 106, and network interface controllers 108 grouped into interconnected nodes. *Id.* ¶ 44. IP blocks 104

represent reusable units of synchronous or asynchronous logic used as building blocks for data processing within NOC 102 such that each IP block 104 is implemented as a generally programmable microcomputer. *Id.*

¶ 53. One of the IP blocks 104 is designated host interface processor (HIP) 128, which provides an interface between NOC 102 and host computer 10. *Id.* ¶ 52. HIP 128 provides data processing services to other IP blocks 104 on NOC 102, including, for example, receiving and dispatching data processing requests from host computer 10 among IP blocks 104. *Id.*

Figure 4 of Kupferschmidt is reproduced below.

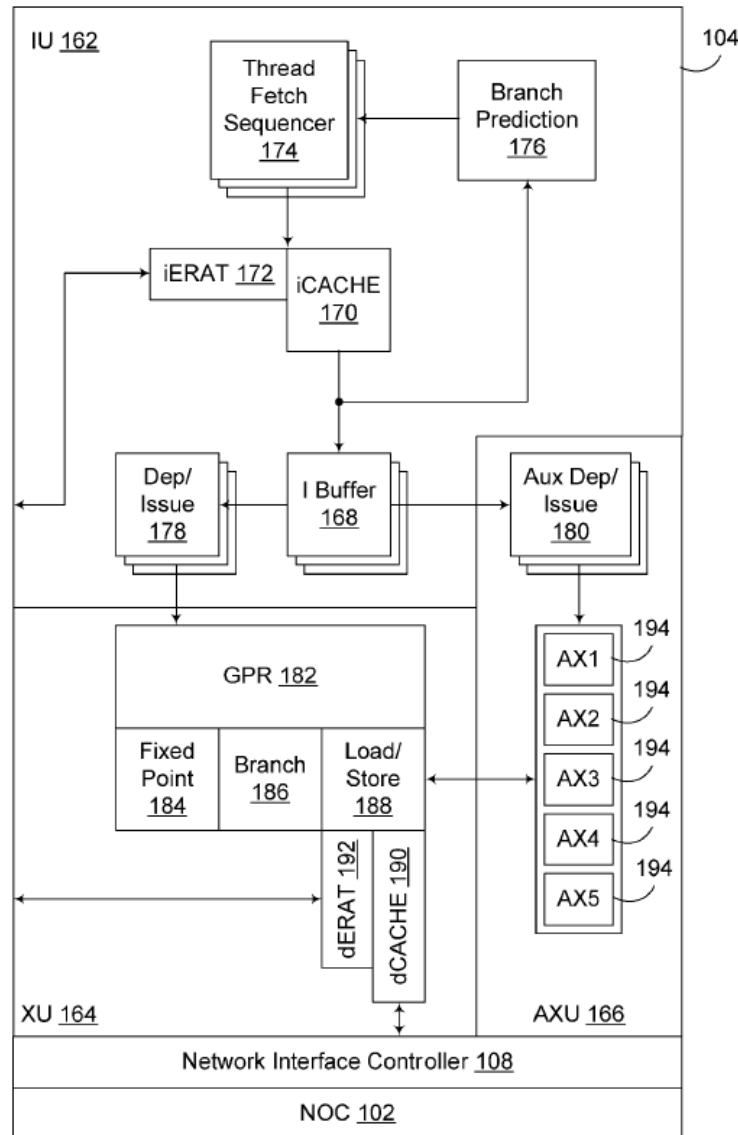


FIG. 4

Figure 4 illustrates IP block 104 implemented as a processing element partitioned into instruction unit (IU) 162, execution unit (XU) 164, and auxiliary execution unit (AXU) 166. Ex. 1007 ¶ 69. IU 162 includes a plurality of thread fetch sequencers 174 and a plurality of instruction buffers 168. *Id.* Each thread fetch sequencer 174 and each instruction buffer 168 is dedicated to a hardware thread. *Id.* Thread fetch

sequencer 174 is used to ensure that instructions to be executed by an associated thread are fetched into L1 instruction cache (iCACHE) 170. *Id.* Instruction buffer 168 receives instructions from iCACHE 170. *Id.* IU 162 also includes dependency/issue logic block 178 dedicated to each hardware thread and configured to resolve dependencies and control the issue of instructions from instruction buffer 168 to execution unit 164. *Id.* ¶ 70. Separate dependency/issue logic block 180 issues instructions concurrently to auxiliary execution unit 166. *Id.*

Figure 5 of Kupferschmidt is reproduced below.

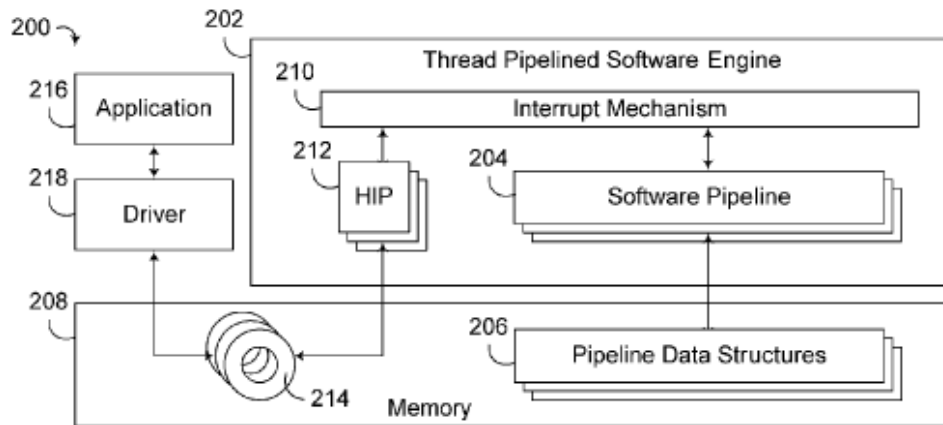


FIG. 5

Figure 5 is a block diagram of thread pipelined software engine 202 suitable for implementation on top of the architecture of NOC 102 in Figure 2.

Ex. 1007 ¶¶ 15, 75. Exemplary processing unit 200 incorporates thread pipelined software engine 202, which is used to implement and execute one or more software pipelines 204. *Id.* ¶ 75. Each pipeline 204 is allocated one or more data structures 206 in shared memory 208 to enable different stages of a pipeline to exchange data. *Id.* Interrupt mechanism 210 enables stages of a pipeline to notify one another of pending work to be performed. *Id.* One or more host interface processors (HIPs) 212 issue work to software

pipelines 204. *Id.* ¶ 76. One or more push buffers 214 are provided to interface each HIP 212 with a software application 216 and driver 218. *Id.* To initiate work in a pipeline, software application 216 issues requests (e.g., application programming interface (API) calls) through driver 218, which generates appropriate requests for HIP 212 and stores the requests in push buffer 214. *Id.* HIP 212 for the relevant pipeline pulls work requests off of push buffer 214 and initiates processing of the request by the associated pipeline. *Id.*

Figure 6 of Kupferschmidt is reproduced below.

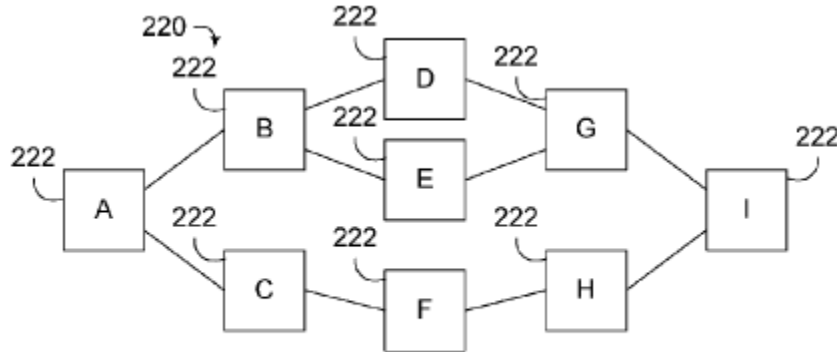


FIG. 6

Figure 6 illustrates exemplary software pipeline 220 including a plurality of stage instances 222 that are separately designated as instances A–I. *Id.* ¶ 80. Each stage instance 222 represents a thread of execution implemented on a particular IP block 104 in NOC 102. *Id.* Stage instances 222 are arranged in pipeline 220 as five stages: a first stage with instance A, a second stage with instances B and C, a third stage with instances D, E, and F, a fourth stage with instances G and H, and a fifth stage with instance I. *Id.* Each stage implements a producer/consumer relationship with the next stage. *Id.* ¶ 79. HIP 212 (shown in Figure 5) sets up the pipeline by configuring each stage with a desired number of instances. *Id.* ¶ 81. Multiple instances may be

assigned to a particular stage to provide additional processing resources relative to other stages so work flows through the pipeline as efficiently as possible and so no single stage presents a bottleneck to performance. *Id.*; *see also id.* ¶ 88 (similar disclosure). In addition, instances may be dynamically added or removed from a stage as needed (based on monitoring the performance of the stages) for balancing the load among the stages of the pipeline. *Id.* ¶ 81; *see also id.* ¶¶ 82, 88 (similar disclosure). Monitored performance statistics may include “time required to complete a data processing task, a number of data processing tasks completed within a particular time period, and so on, as will occur to those of skill in the art.” *Id.* ¶ 82.

2. *Du*

Du is a U.S. patent application publication directed to a multimedia processor. Ex. 1008, code (57), ¶ 2. Figure 1 of *Du* is reproduced below.

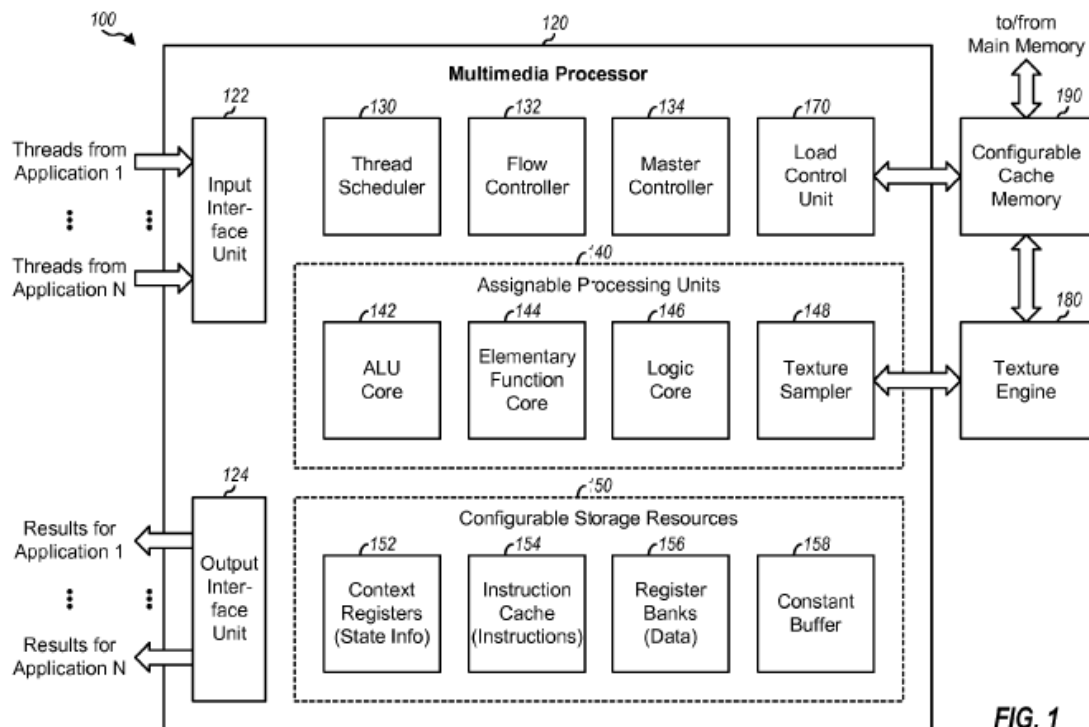


FIG. 1

Figure 1 depicts multimedia system 100, which supports multimedia applications 1–N. *Id.* ¶ 19. Multimedia system 100 includes multimedia processor 120, which performs processing for the N applications and may split the processing of each application into a series of execution threads (i.e., specific tasks that may be performed with a set of one or more instructions). *Id.* ¶¶ 21–22. Multimedia processor 120 includes, *inter alia*, thread scheduler 130, which performs various functions to schedule and manage execution of the threads. *Id.* ¶ 23.

Figure 6 of Du is reproduced below.

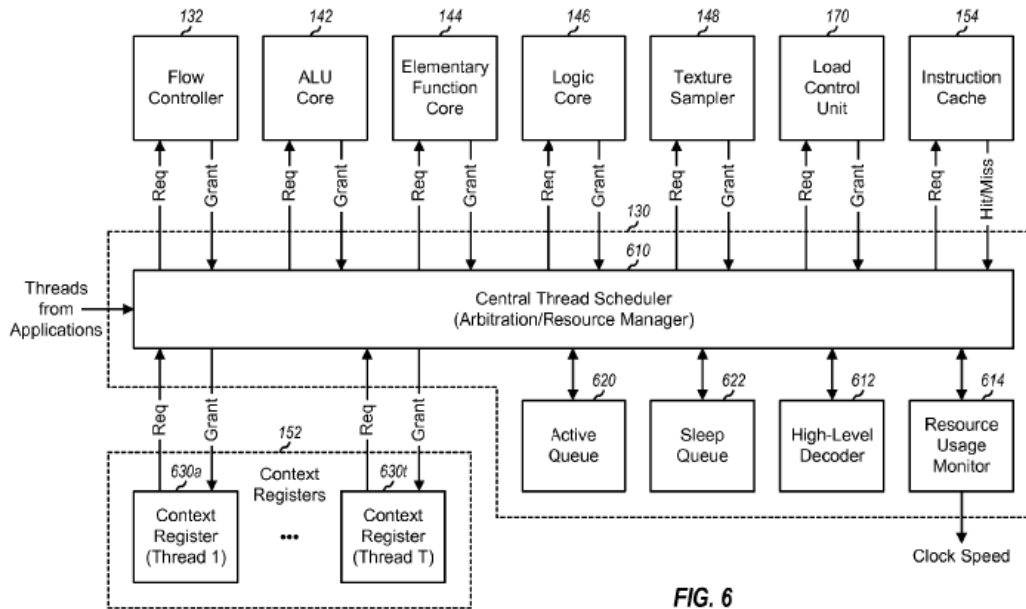


FIG. 6

Figure 6 is a block diagram of thread scheduler 130. *Id.* ¶ 42. Thread scheduler 130 includes central thread scheduler 610, which performs various functions to schedule threads and manages execution of threads. *Id.* ¶¶ 44–45. Central thread scheduler 610 may place a thread in active queue 620 when required resources are available or sleep queue 622 when resources are not available. *Id.* ¶ 44. Central thread scheduler 610 may also determine the processing units to use for various threads, check for

read/write conflicts of storage units, and dispatch different threads to different processing units for execution. *Id.* ¶ 45.

3. Anderson

Anderson is a U.S. patent application publication directed to “the control of priorities and instruction issue rates on a multithreaded processor.” Ex. 1009 ¶ 1. Figure 2 of Anderson is reproduced below.

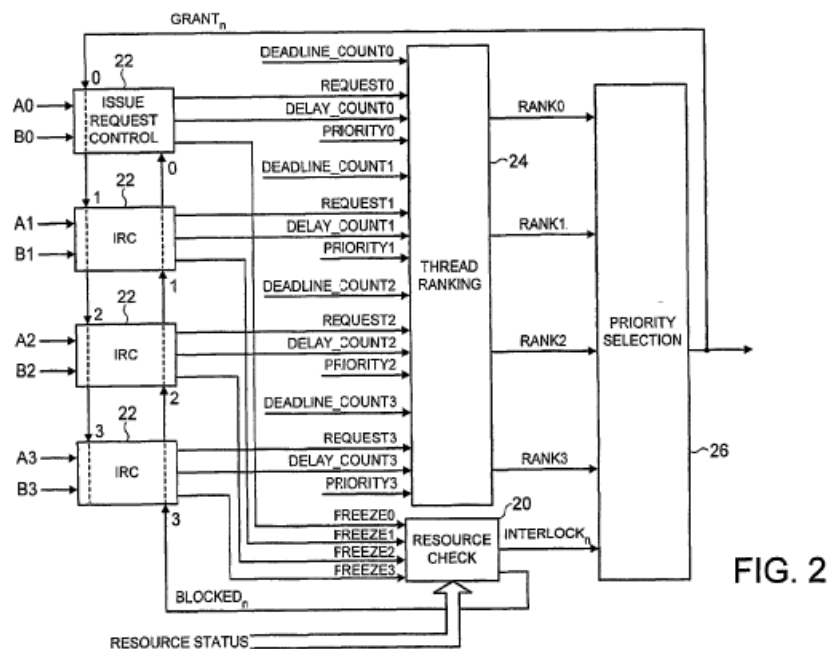


Figure 2 is a block diagram of the thread scheduling portion of a media control core (MCC), i.e., a fine-grained multithreaded processor. *Id.* ¶¶ 16, 38. Resource checker 20 receives resource status information relating to the status of data processing units, pipeline and memory bank status, and input/output port status. *Id.* ¶ 38. Thread ranking unit 24 sends out ranking signals for each instruction thread to priority selection unit 26. *Id.* ¶ 38. Priority control selection unit 26 provides a rank order of preference for the threads. *Id.* ¶ 75.

4. *Claim 8*

a. *Petitioner's Contentions*

The preamble of claim 8 recites “[a] method for managing execution of a plurality of instances of a program on a parallel processing system comprising an array of processor cores and a control system.” Ex. 1001, 22:31–33. Petitioner cites Kupferschmidt’s NOC 102 for teaching the recited “parallel processing system” and contends NOC 102 comprises an array of IP blocks 104 (i.e., the recited “array of processor cores”). Pet. 52–55 (citing, *inter alia*, Ex. 1007 ¶¶ 7, 26, 41, 44, 46, 68–81, 88, Figs. 2, 5, 6). According to Petitioner, “[t]he set of stage-instances assigned to the hardware threads on a single, multi-threaded IP block 104 provides a ***‘plurality of instances of a program.’***” *Id.* at 55 (citing Ex. 1003 ¶¶ 148–149). For the recited “control system,” Petitioner proposes modifying Kupferschmidt’s IP blocks 104 “in accordance with Du’s suggestion for classifying threads in active/sleep states based on whether the resources required by the thread are currently available for execution.” *Id.* at 47 (citing Ex. 1003 ¶¶ 135–136), 57 (citing Ex. 1003 ¶ 152). In particular, Petitioner cites Du’s teaching that central thread scheduler 610 classifies threads as “active” or “sleeping” based on whether resources are available and places them accordingly in active queue 620 or sleep queue 622. *Id.* at 46–47 (citing Ex. 1008 ¶¶ 42–54, Figs. 1, 2, 4–6). In this way, scheduling logic in Kupferschmidt’s IP block 104 could “defer issuing instructions for a thread to an execution unit until input data for messages to be processed by the thread are loaded into local cache at the IP block 104.” Pet. Reply 24 (citing Ex. 1003 ¶¶ 131–139; Pet. 44–50). Similarly, the scheduling logic could also “wait for instructions to become available in a local cache before

attempting to issue instructions.” *Id.* at 24–25 (citing Ex. 1003 ¶¶ 131–139; Ex. 1007 ¶ 69; Ex. 1023 ¶¶ 43–45; Pet. 44–50).

Petitioner notes that Kupferschmidt’s IP block 104 “would ordinarily require some logic for selecting threads or instructions to issue to the execution units at each time slot,” but “Kupferschmidt offers little detail in this regard.” Pet. 47 (citing Ex. 1003 ¶ 136); *see also* Pet. Reply 23–24 (citing Ex. 1003 ¶¶ 135–139; Ex. 1023 ¶¶ 42–43) (similar argument). Specifically, Petitioner contends “Kupferschmidt does not explain . . . when logic blocks 178/180 would begin issuing instructions for a thread and whether instructions would always begin issuing as soon as they arrive in the instruction buffer 168 or if other conditions may require satisfaction.” Pet. Reply 24; Ex. 1023 ¶ 44. As such, Petitioner contends an ordinarily skilled artisan would have modified Kupferschmidt’s IP block 104 to incorporate logic implementing Du’s active/sleep thread classifications in order “to provide more efficient use of processing resources on an IP block 104.” Pet. 48–49 (citing Ex. 1003 ¶¶ 138–139; Ex. 1008 ¶¶ 23, 31, 42–49, Figs. 1–2, 6). Petitioner contends that the resulting system would “better ensure that a thread could execute successfully when its instructions begin issuing to an execution unit 164/166 on the IP block 104,” thus “beneficially reduc[ing] dead time on the execution units and minimiz[ing] stalls that could otherwise occur.” *Id.* at 49–50 (citing Ex. 1003 ¶ 139; Ex. 1008 ¶¶ 44–46).

Petitioner proposes a further modification of the Kupferschmidt–Du combination based on Anderson for teaching recited “control logic.” *See* Pet. 50–52. Specifically, Petitioner cites Anderson’s teaching of “techniques for ranking hardware threads on a multi-threaded processor, and

control[ling] instruction issuance to an execution unit based on the respective rankings of the various hardware threads.” *Id.* at 50. According to Petitioner, Anderson teaches a processor with thread ranking unit 24 that ranks hardware threads based on a number of metrics, such as “thread priority.” *Id.* at 43–44 (citing Ex. 1009 ¶¶ 38–39, 70, 77–79, 87, Fig. 2), 50 (citing Ex. 1009 ¶¶ 23–24, 30–35, 78, 87–95, Fig. 2). Petitioner contends that “it would have been obvious to apply Anderson’s teachings to further modify Kupferschmidt and Du in view of Anderson such that active hardware threads on an IP block would be ranked, e.g., based on metrics such as ‘thread priority’ and, optionally, one or more additional metrics.” *Id.* at 50–51 (citing Ex. 1009 ¶ 140). Thus, in Petitioner’s proposed combination with Anderson, a component of Kupferschmidt’s thread scheduler “would control the issuance of instructions from the active threads selected for execution using the respective priority rankings of the threads.” *Id.* at 51 (citing Ex. 1003 ¶ 140; Ex. 1009 ¶¶ 30–35, Fig. 2). Petitioner contends that an ordinarily skilled artisan would have made the combination “to realize known benefits associated with hardware thread prioritization,” such as differentiating the priority of multiple ready-to-run hardware threads relative to one another. *Id.* (citing Ex. 1003 ¶ 141).

Claim 8 further recites “classifying the plurality of instances into a set of priority classes based at least in part on determining, for each instance of the plurality of instances.” Ex. 1001, 22:35–37. Petitioner contends that “scheduling logic (*control system*) in an IP block 104 (*processor core*) of a Kupferschmidt-Du-Anderson system is configured to classify hardware threads into ‘active’ or ‘sleep’ states based on the availability of resources required to execute the thread.” Pet. 58 (citing Ex. 1003 ¶¶ 154–160;

Ex. 1007 ¶¶ 69–74; Ex. 1008 ¶¶ 42–46, Fig. 6). Petitioner further contends that,

[b]y classifying the hardware threads hosted on an IP block 104 into active/sleep states, scheduling logic in the IP block also classifies the set of stage-instances (***plurality of instances***) assigned to the hardware threads of the IP block into the active/sleep states, due to the 1:1 correspondence established between stage-instances and their assigned hardware threads.

Id. at 58–59 (citing Ex. 1003 ¶ 155; Ex. 1007 ¶¶ 26, 69, 75–85, 99, Figs. 5–6). Petitioner also cites Du’s teaching of selecting hardware threads for execution based on active/sleep status (*see id.* (citing Ex. 1008 ¶¶ 44–46)) and contends that, in light of Anderson, such selection could also involve ranking “active” threads to define priorities for scheduling and execution (*see id.* at 59–60 (citing Ex. 1009 ¶¶ 76–80, 87–95)). Thus, according to Petitioner, “[t]he Kupferschmidt-Du-Anderson combination . . . classifies threads into a ‘***set of priority classes***’ that includes a ‘sleep’ state priority class, and multiple ‘active’ state priority classes that correspond to different thread ranking or priority levels.” *Id.* at 60.

Claim 8 further recites that the “classifying” step is “based at least in part on determining, for each instance of the plurality of instances, (i) whether the respective instance is waiting for arrival of input data at one or more input buffers of the respective instance.” Ex. 1001, 22:36–40. Petitioner cites Du’s teaching that central thread scheduler 610 determines whether “resources . . . required by [a] thread are available” before deeming the thread ready for execution. Pet. 62 (citing Ex. 1008 ¶ 44). According to Petitioner,

the Kupferschmidt-Du-Anderson scheduling logic (***control system***) would wait for arrival of at least one message (***input data***) at one or more inboxes (***one or more input buffers***) of the

respective hardware thread (*instance*) as a condition for classifying the thread into an “active” priority class (otherwise the thread would be classified in a “sleep” state).

Id. at 63–64 (citing Ex. 1003 ¶ 163; Ex. 1007 ¶¶ 8–9, 29–32, 75–82, 97–100, Figs. 5–6, 11–12) (footnote omitted). As such, Petitioner explains that the “ready for execution” determination encompasses determining “that a message (*input data*) had arrived in a dedicated inbox (*input buffer*) of the stage instance or its corresponding hardware thread.” *Id.* at 64 (citing Ex. 1003 ¶¶ 163–164).

Claim 8 further recites that the “classifying” step is “based at least in part on determining, for each instance of the plurality of instances, . . . (ii) whether the respective instance is waiting for completion of memory content transfers to update one or more fast-access memories of the respective instance.” Ex. 1001, 22:36–37, 22:41–44. Petitioner maps the recited “fast-access memories” to Kupferschmidt’s instruction buffers 168 in instruction unit 162 of IP block 104; Petitioner notes that “instruction buffers 168 . . . receive instructions from an L1 instruction cache (iCACHE) 170.” Pet. 66 (quoting Ex. 1007 ¶ 69). Petitioner further notes that “the corresponding dependency/issue logic block 178/180 for each thread issues instructions from that thread’s instruction buffer 168 to an appropriate execution unit 164/166.” *Id.* (citing Ex. 1007 ¶¶ 69–74, Fig. 4).

Petitioner contends that it would have been obvious to apply Du’s teachings about a “ready for execution” determination “such that scheduling logic (*control system*) on the IP block in the combination would classify hardware threads into one of the active priority classes or a sleep class based on the availability of instructions in a respective instruction buffer (*fast-access memory*) of the hardware thread on the IP block.” *Id.* at 66–67

(citing Ex. 1003 ¶¶ 170–172). Petitioner further contends that an ordinarily skilled artisan “would have sought to condition a hardware thread’s execution readiness (e.g., active/sleep state) on the availability of instruction(s) in buffers 168—particularly since buffers 168 provide the memories most closely coupled to the IP block’s execution units.” *Id.* at 67 (citing Ex. 1003 ¶¶ 170–172; Ex. 1007, Fig. 4).

Claim 8 further recites “selecting a subset of the plurality of instances for execution on a subset of the array of processor cores based at least in part based on the classifying.” Ex. 1001, 22:45–47. Petitioner contends that when

a stage-instance receives a new message for processing . . . , and once all other conditions have been met for the thread to become “ready to execute,” scheduling logic in a Kupferschmidt-Du-Anderson IP block would re-classify the thread from the “sleep” state to one of the prioritized “active” states.

Pet. 71 (citing Ex. 1008 ¶¶ 44–46). According to Petitioner, this results in Kupferschmidt’s scheduler “select[ing] the thread for the activated stage-instance (*instance*) for execution, e.g., by issuing instructions from the activated thread to execution units 164 and/or 166.” *Id.* (citing Ex. 1007 ¶¶ 69–73; Ex. 1008 ¶¶ 44–46). Petitioner contends that a subset of hardware threads (i.e., the recited “instances”)—only the active ones—are selected for execution. *Id.* at 72 (citing Ex. 1003 ¶ 178). Petitioner further contends that threads are selected for execution on a particular IP block 104 (i.e., the recited “processor core”) among an array of IP blocks 104, thus meeting “the subset of the array of processor cores” limitation. *Id.* (citing Ex. 1003 ¶ 178; Ex. 1007, Fig. 2).

Claim 8 further recites that “at least one of the classifying and the selecting is performed by the control system.” Ex. 1001, 22:48–49. Petitioner contends that, in its mapping of claim 8, it relies on classifying and selecting of threads for execution in the scheduling logic of Kupferschmidt’s IP block 104. Pet. 73 (citing Ex. 1003 ¶ 179; Ex. 1007 ¶¶ 69–73; Ex. 1008 ¶¶ 44–46, Fig. 6). Petitioner notes that it maps the recited “control system” to the scheduling logic. *Id.*

b. Patent Owner’s Arguments

Patent Owner disputes Petitioner’s rationale for combining Du with Kupferschmidt, which Patent Owner characterizes as “illusory.” PO Resp. 37. Specifically, Patent Owner argues that “grafting on Du’s active vs. sleep state classification would be wholly unnecessary because Kupferschmidt was already designed such that instructions would not issue until data was ready to process.” PO Sur-reply 21. In support of this argument, Patent Owner cites Dr. Chase’s cross-examination testimony “that Kupferschmidt would already be designed such that a thread would not become active and execute until it had the required input data and instructions.” *Id.* (citing Ex. 2003, 113:3–19); PO Sur-reply 22 (quoting same).

Patent Owner contends that Kupferschmidt already addresses the problem of stalls and dead time through a priority selection process and load balancing. PO Resp. 21 (citing Ex. 2002 ¶ 64). Patent Owner cites Kupferschmidt’s teachings of priority and non-priority inboxes for a hardware thread into which messages from other threads flow. *Id.* at 23 (citing Ex. 1007 ¶¶ 7–8; Ex. 2002 ¶¶ 66, 68). Patent Owner also notes that

messages themselves are associated with a priority. *Id.* (citing Ex. 1007 ¶ 28; Ex. 2002 ¶ 68). Patent Owner additionally cites host interface processors (HIPs) within Kupferschmidt’s thread pipelined software engine as managing the software pipeline and assigning stage instances to hardware threads. *Id.* at 24–27 (citing Ex. 1007 ¶¶ 81–82, 88; Ex. 2002 ¶¶ 76–78). Patent Owner explains that Kupferschmidt’s “HIP is able to provide stage instances with additional hardware resources as needed to avoid data stalls” and “to allocate additional hardware threads to stage instances to avoid bottlenecks.” *Id.* at 25 (citing Ex. 2002 ¶ 76).

c. Analysis

In light of Patent Owner’s arguments, we are not persuaded by Petitioner’s reasons for combining Du with Kupferschmidt. Both Dr. Chase and Dr. Smith agree that Kupferschmidt already is designed so that threads do not become active (i.e., execute) until all needed resources are available. Ex. 2002 ¶¶ 76–78, 98; Ex. 2003, 113:3–19. Referring to the “notion of a thread being forced to wait to issue instructions because resources are not available,” Dr. Chase testified in his deposition that “a person of skill in the art would understand that that’s present as for any multithreaded architecture.” Ex. 2003, 114:19–24. Dr. Chase also testified that Du’s disclosure of a “logical unit for classifying threads as active or sleeping based on the availability of resources” is “consistent, in [his] opinion, with how a person of skill in the art would understand Kupferschmidt.” *Id.* at 114:2–8. Furthermore, Dr. Smith testifies that Kupferschmidt’s “HIP is able to provide a given stage instance with additional hardware resources as

needed to ensure that data moves efficiently through the system and without bottlenecks and data stalls.” *Id.* ¶ 76 (quoting Ex. 1007 ¶ 88).

Indeed, Petitioner’s own characterizations of Kupferschmidt show that Kupferschmidt already includes functionality akin to Du’s active and sleeping classifications. For example, Petitioner states that, “after Kupferschmidt’s host interface processor (HIP) assigns stage-instances to hardware threads, a stage instance/hardware thread may remain idle while waiting for input data to arrive for processing in a dedicated inbox (and may also be idle after processing one message and waiting for another).” Pet. 42 (citing Ex. 1003 ¶ 66; Ex. 1007 ¶¶ 81, 88); *see also id.* at 70 (citing Ex. 1007 ¶¶ 80, 97–99, Fig. 6, for the proposition that “later stages in software pipeline must wait to receive inputs from earlier stages”). Petitioner also states that, “[w]hen an input is available, the stage instance for the hardware thread may execute to perform operations on the input date [sic, data].” Pet. 42 (citing Ex. 1003 ¶ 66; Ex. 1007 ¶¶ 81, 88). Petitioner’s characterizations support Dr. Smith’s testimony that Kupferschmidt already is designed to function such that “threads do not begin to execute until instructions and messages are present at Kupferschmidt’s inboxes and local IP block buffers.” Ex. 2002 ¶ 98.

This existing functionality within Kupferschmidt’s system undermines Petitioner’s asserted reasons for combining Du with Kupferschmidt. In particular, Petitioner contends that an ordinarily skilled artisan “would have found Du’s active/sleep classifications beneficial to provide more efficient use of processing resources on an IP block 104.” Pet. 49 (citing Ex. 1003 ¶ 139). According to Petitioner, the efficiency gains arise from reduced dead time and minimized stalls. *Id.* at 49–50 (citing Ex. 1003 ¶ 139). Yet

Petitioner ascribes these efficiency gains to the same functionality in Du that exists in Kupferschmidt, i.e., “conditioning thread activation on the availability of resources necessary to execute (including input data and instruction availability)” and that “threads that were not ready to execute . . . would be prevented from having instructions issued to an execution unit ahead of other thread(s) that were already available to execute.” *Id.* at 49–50 (citing Ex. 1003 ¶ 139; Ex. 1008 ¶¶ 44–46). Because Du’s added functionality is not discernably different from Kupferschmidt’s existing functionality, we are not persuaded that adding Du’s active/sleep classifications to Kupferschmidt’s system would result in the efficiency gains touted by Petitioner. *See* Ex. 2002 ¶ 92 (“[T]he alleged benefits of making the combination are already adequately addressed in the Kupferschmidt system as originally disclosed.”). Further, to the extent Petitioner attempts to distinguish Du as conditioning thread activation based on the availability of hardware resources rather than just data or instructions (*see, e.g.*, Ex. 1023 ¶ 46 n.3), we again are not persuaded that Du would result in further efficiency gains over Kupferschmidt’s existing functionality. Kupferschmidt’s HIP monitors workflow and performance, dynamically reallocates hardware threads, and issues work requests, which evidences that Kupferschmidt’s system already considers the availability of hardware resources. Ex. 1007 ¶¶ 81–82, 88. Indeed, Dr. Chase testified on cross-examination that threads in Kupferschmidt are not activated until *all* needed resources are available. Ex. 2003, 113:3–19.

We also have considered Petitioner’s arguments about alleged gaps in Kupferschmidt such as “Kupferschmidt’s exceedingly limited and terse description of the IP block’s dependency/issue logic blocks 178/180.” Pet.

Reply 23–24 (citing Ex. 1023 ¶¶ 42–43); *see also* Pet. 47 (citing Ex. 1003 ¶ 136) (similar argument). First, as stated above, we do not perceive a substantive gap in Kupferschmidt’s teachings that is filled by Du. At most, Petitioner has shown a gap in Kupferschmidt’s “vocabulary” regarding “threads that can’t be selected for execution because the resources they require are not available.” Ex. 2003, 114:9–13, 114:25–115:2; *see also* Ex. 1023 ¶ 46 n.3 (“Du *formalizes* the notion of active/sleep state classifications” (emphasis added)). But this does not support a gap in the substantive teaching of Kupferschmidt, as discussed above. Second, even if there were a substantive gap in Kupferschmidt’s express disclosure, it would still not cure the faults in Petitioner’s rationale for the combination, because Petitioner has not persuasively shown that applying Du’s teachings to Kupferschmidt would result in the benefits Petitioner asserts.

Finally, we note that Petitioner seemingly acknowledges that Du’s teachings are unnecessary in its proposed combination. Petitioner states that, “[e]ven if . . . Kupferschmidt’s original design implicitly includes active/sleep state classifications as taught in Du, the combination would still render the claims obvious” based on “the relevant teachings from Kupferschmidt.” Pet. Reply 23 n.6.⁷ But we cannot depart from the particular ground including Du that Petitioner originally asserted, because

⁷ In addition to this footnote in the Reply, Dr. Chase notes that Kupferschmidt does not “expressly disclose[] active/sleep state classifications as taught in Du” and “that such classifications are [not] inherent in Kupferschmidt.” Ex. 1023 ¶ 46 n.3. To be clear, the issue of whether such classifications are expressly or inherently disclosed in Kupferschmidt is not the basis of our decision. Rather, our conclusions rest on Petitioner’s faulty rationale for combining Du with Kupferschmidt.

“the petitioner’s petition, not the Director’s discretion, is supposed to guide the life of the litigation.” *SAS Inst., Inc. v. Iancu*, 138 S. Ct. 1348, 1356 (2018); *see also Sirona Dental Sys. GmbH v. Institut Straumann AG*, 892 F.3d 1349, 1356 (Fed. Cir. 2018) (“It would . . . not be proper for the Board to deviate from the grounds in the petition and raise its own obviousness theory.”). Petitioner is master of its Petition, so we evaluate the ground including Du asserted by Petitioner. And, as explained above, we are not persuaded by Petitioner’s rationale for combining Du with Kupferschmidt.

Thus, regarding claim 8, we determine that Petitioner has not shown, by a preponderance of the evidence, that the claimed invention as a whole would have been obvious over the combination of Kupferschmidt, Du, and Anderson.

5. *Claims 13, 21, and 24*

Petitioner’s analysis for claims 13, 21, and 24 incorporates the same faulty rationale for combining Du with Kupferschmidt that we have discussed above. *See supra* § II.F.4. Thus, for each of claims 13, 21, and 24, we determine that Petitioner has not shown, by a preponderance of the evidence, that the claimed invention as a whole would have been obvious over the combination of Kupferschmidt, Du, and Anderson.

G. *Obviousness Ground Based on Kupferschmidt, Du, Anderson, and Daly*

Petitioner contends claims 9–12, 14, 22, and 23 would have been obvious over the combination of Kupferschmidt, Du, Anderson, and Daly. Pet. 76–88; Pet. Reply 15–26. Patent Owner disputes Petitioner’s contentions. PO Resp. 21–40; PO Sur-reply 14–27.

Petitioner's analysis for claims 9–12, 14, 22, and 23 incorporates the same faulty rationale for combining Du with Kupferschmidt that we have discussed above. *See supra* § II.F.4. Thus, for each of claims 9–12, 14, 22, and 23, we determine that Petitioner has not shown, by a preponderance of the evidence, that the claimed invention as a whole would have been obvious over the combination of Kupferschmidt, Du, Anderson, and Daly.

III. CONCLUSION⁸

Petitioner has shown, by a preponderance of the evidence, that (1) claim 1 would have been obvious over the combination of Stevens and Smetana; and (2) claim 2 would have been obvious over the combination of Stevens, Smetana, and Daly. Petitioner, however, has not shown, by a preponderance of the evidence, that (3) claims 8, 13, 21, and 24 would have been obvious over the combination of Kupferschmidt, Du, and Anderson; and (4) claims 9–12, 14, 22, and 23 would have been obvious over the combination of Kupferschmidt, Du, Anderson, and Daly.

⁸ Should Patent Owner wish to pursue amendment of the challenged claims in a reissue or reexamination proceeding subsequent to the issuance of this decision, we draw Patent Owner's attention to the April 2019 *Notice Regarding Options for Amendments by Patent Owner Through Reissue or Reexamination During a Pending AIA Trial Proceeding*. *See* 84 Fed. Reg. 16,654 (Apr. 22, 2019). If Patent Owner chooses to file a reissue application or a request for reexamination of the challenged patent, we remind Patent Owner of its continuing obligation to notify the Board of any such related matters in updated mandatory notices. *See* 37 C.F.R. § 42.8(a)(3), (b)(2).

IV. ORDER

Accordingly, it is

ORDERED that claims 1 and 2 of the '353 patent are unpatentable;

ORDERED that claims 8–14 and 21–24 of the '353 patent are not unpatentable; and

FURTHER ORDERED that, because this is a Final Written Decision, parties to this proceeding seeking judicial review of our decision must comply with the notice and service requirements of 37 C.F.R. § 90.2.

In summary:

Claim(s)	35 U.S.C. §	Reference(s)/Basis	Claim(s) Shown Unpatentable	Claim(s) Not shown Unpatentable
1	103(a)	Stevens, Smetana	1	
2	103(a)	Stevens, Smetana, Daly	2	
8, 13, 21, 24	103(a)	Kupferschmidt, Du, Anderson		8, 13, 21, 24
9–12, 14, 22, 23	103(a)	Kupferschmidt, Du, Anderson, Daly		9–12, 14, 22, 23
Overall Outcome			1, 2	8–14, 21–24

IPR2022-00574
Patent 10,318,353 B2

PETITIONER:

Walter Renner
David Holt
Nicholas Stephens
FISH & RICHARDSON P.C.
axf-ptab@fr.com
holt2@fr.com
nstephens@fr.com

PATENT OWNER:

Scott McKeown
ROPES & GRAY LLP
scott.mckeown@ropesgray.com

W. Cook Alciati
Michael Dorfman
GARDELLA GRACE P.A.
calciati@gardellagrace.com
mdorfman@gardellagrace.com